HeteroSync: A Benchmark Suite for Fine-Grained Synchronization on Tightly Coupled GPUs

Matthew D. Sinclair, Johnathan Alsop, Sarita V. Adve
University of Illinois @ Urbana-Champaign
hetero@cs.illinois.edu
Traditional Heterogeneous SoC Memory Hierarchies

Discrete address spaces
Works well for streaming applications
Inefficient for applications with fine-grained synchronization
Motivation

- Tighter CPU-GPU integration – need better synch support
- Lots of heterogenous coherence, consistency research

QuickRelease HPCA’14
DeNovo MICRO ‘15
hLRC MICRO ‘16
RAts ISCA ’17

HRF ASPLOS ‘14
RemoteScopes ASPLOS ‘15
hVIPS TACO ‘16
...

No standardization – which approach is best?

HeteroSync: new microbenchmark suite
• Fine-grained synchronization microbenchmarks
  – Various mutex, semaphore, barrier algorithms
  – Relaxed atomics: event counters, split counters, seqlocks, ...

• Enable deep analysis of:
  – Algorithm scalability
  – Scalability of different coherence and consistency schemes

Standard fine-grained synch microbenchmarks
• Motivation
• **Background: Coherence & Consistency**
• HeteroSync
• Results
• Conclusion
Atomics Background

• Default: Data-race-free-0 (DRF0) [Adve ISCA ‘90]
  – Identify all races as synchronization accesses (C++: atomics)

  // each thread
  for i = 0:n
  ...
  ...
  ADD R4, A[i], R1  synch (atomic)
  ADD R5, B[i], R1  synch (atomic)
  ...

  – All atomics order data accesses
  – Atomics order other atomics

⇒ Ensures SC semantics if no data races
Atomics Background (Cont.)

• Default: Data-race-free-0 (DRF0) [Adve ISCA ‘90]
  – All atomics order data
  – All atomics order other atomic accesses
  \[\Rightarrow\text{Ensures SC semantics if no data races}\]

• Relaxed atomics [Boehm PLDI ‘08]
  + Do not order data or other atomics
  \[\Rightarrow\text{But can violate SC and no formal specification}\]

• Data-race-free-relaxed (DRFrlx) [Sinclair ISCA ‘17]
  \[\Rightarrow\text{SC-centric semantics + efficiency}\]
GPU Coherence with DRF

- With data-race-free (DRF) memory model
  - No data races; synchs must be explicitly distinguished
  - Synchronization accesses (atomics) go to last level cache (LLC)
  - Synchronization points are expensive, preclude reuse

Simple but inefficient coherence, simple consistency
• New memory model: Heterogeneous-race-free (HRF) [ASPLOS ‘14]
  – Adds scoped synchronization

```c
// each thread
for i = r[tid]:r[tid+1]
  LOCK   global
  LD R1, A[i];
  LD R2, B[i];
  R3 \leftarrow Math(R1, R2);
  ST B[i], R3;
  UNLOCK global
```
• New memory model: Heterogeneous-race-free (HRF)
  – Adds scoped synchronization
  – No overhead for locally scoped synchronizations
• But higher programming complexity
  More efficient coherence, complex consistency
DeNovo Coherence with DRF

- Reuse dirty data across synch points – more data reuse
- Synchronization accesses can be performed at L1 – synch reuse

Efficient coherence, simple consistency

for $i = r[tid]:r[tid+1]$

LOCK
LD $R1, A[i]$;
LD $R2, B[i]$;
$R3 \leftarrow \text{Math}(R1, R2)$;
ST $B[i], R3$;
UNLOCK
Outline

• Motivation
• Background: Coherence & Consistency
  • HeteroSync
    – Synchronization Primitives Microbenchmarks
    – Relaxed Atomics Microbenchmarks
• Results
• Conclusion
SyncPrims microbenchmarks [Stuart CoRR ’11]:
- Originally studied synchronization primitive latency
- Focus: performance of atomic operations
- Less Focus: overheads of proper synchronization
  - No global data accesses

**Microbenchmarks:**
- Mutexes: Spin (with backoff), centralized ticket, ring buffer
- Semaphores: Spin (with backoff)
- Barriers: Centralized, decentralized barriers
Synchronization Primitives Microbenchmarks

• Updates [Sinclair MICRO ‘15]:
  – Global data accesses in critical sections
  – Synchronization loads and stores to enforce ordering
  – Two versions of each microbenchmark: local/global scope
  – Optimize algorithms

• Microbenchmarks:
  – Mutexes: Spin (with backoff), centralized ticket, ring buffer
decentralized ticket
  – Semaphores: Spin (with backoff)
  – Barriers: Centralized, decentralized barriers
    2-level tree + local exchange
  Can vary data size, scope, synchronization primitive
Outline

• Motivation
• Background: Coherence & Consistency
  • HeteroSync
    – Synchronization Primitives Microbenchmarks
    – Relaxed Atomics Microbenchmarks
• Results
• Conclusion
• Contacted vendors, developers, and researchers
  – Common uses of relaxed atomics [Sinclair ISCA ‘17]:

<table>
<thead>
<tr>
<th>Feature</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Event Counters</td>
<td>Place events into bins</td>
</tr>
<tr>
<td>Seqlocks</td>
<td>Sequence number instead of mutex lock</td>
</tr>
<tr>
<td>Flags</td>
<td>Shared flag for inter-thread communication</td>
</tr>
<tr>
<td>Split Counters</td>
<td>Simultaneously update and get partial sums</td>
</tr>
<tr>
<td>Ref Counters</td>
<td>Track threads using an object; delete if none</td>
</tr>
</tbody>
</table>

**Can vary data size, algorithm**
Outline

• Motivation
• Background: Coherence & Consistency
• HeteroSync
• Results
• Conclusion
Evaluation Methodology

• 1 CPU core + 1-15 GPU compute units (CU)
  – Each node has private L1, scratchpad, tile of shared L2

• Simulation Environment
  – GEMS, Simics, Garnet, GPGPU-Sim

• HeteroSync microbenchmarks
  – SyncPrims: weak scaling
  – Relaxed Atomics: strong scaling
<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Coherence</th>
<th>Consistency</th>
</tr>
</thead>
<tbody>
<tr>
<td>GD0</td>
<td>GPU</td>
<td>DRF0</td>
</tr>
<tr>
<td>DD0</td>
<td>DeNovo</td>
<td>DRF0</td>
</tr>
<tr>
<td>GDR</td>
<td>GPU</td>
<td>DRFrlx</td>
</tr>
<tr>
<td>DDR</td>
<td>DeNovo</td>
<td>DRFrlx</td>
</tr>
</tbody>
</table>

Configurations Studied

Studied GPU, DeNovo coherence with DRF0, DRFrlx, HRF
Key Evaluation Questions

• How are coherence/consistency schemes impacted?
  – Do certain algorithms scale better than others?
  – How does an algorithm scale with local/global scope?
  – Do relaxed atomics impact scalability?
As CUs increase, execution time increases due to increased contention. DeNovo+DRF is able to reuse synch, so scales 20% better than GPU+HRF.
As CUs increase, execution time increases due to increased contention.

Decentralized ticket lock scales better than centralized with DeNovo+DRF.

For decentralized, DeNovo+DRF scales 32% better than GPU+HRF.
Key Evaluation Questions

- How are coherence/consistency schemes impacted?
  - Do certain algorithms scale better than others?

Coherence protocol impacts which algorithm scales better

- How does an algorithm scale with local/global scope?
- Do relaxed atomics impact scalability?
Decentralized Ticket Lock Scalability (Local Scope)

- GPU+DRF cannot perform atomics locally, contention increases with # CUs
- DeNovo+DRF exploits locality
GPU+DRF cannot perform atomics locally, contention increases with # CUs
DeNovo+DRF exploits locality
GPU+HRF also exploits locality, but increased programming complexity
Key Evaluation Questions

• How are coherence/consistency schemes impacted?
  – Do certain algorithms scale better than others?
  
  **Coherence protocol impacts which algorithm scales better**
  – How does an algorithm scale with local/global scope?
  **DeNovo+DRF provides best scalability with global scope**
  **GPU+HRF and DeNovo+DRF both scale well with local scope**
  – Do relaxed atomics impact scalability?
Execution time significantly reduced as work spread across more CUs
DeNovo+DRF0: tradeoff between increased reuse, remote accesses
Execution time significantly reduced as work spread across more CUs. DeNovo+DRF0: tradeoff between increased reuse, remote accesses. Relaxed atomics reduce execution time compared to DRF0.
Key Evaluation Questions

• How are coherence/consistency schemes impacted?
  – Do certain algorithms scale better than others?
    Coherence protocol impacts which algorithm scales better
  – How does an algorithm scale with local/global scope?
    DeNovo+DRF provides best scalability with global scope
    GPU+HRF and DeNovo+DRF both scale well with local scope
  – Do relaxed atomics impact scalability?
    Relaxed atomics reduce execution time, but increase contention

  Compare schemes and scalability with HeteroSync
• **HeteroSync**: fine-grained GPU synch microbenchmarks
  – Synchronization primitives: mutexes, semaphores, barriers
  – Relaxed atomics: event counters, split counters, seqlocks, …
  – Highly configurable

• Study algorithms, coherence, and consistency
  – Examine scalability of existing approaches

• Standard set of GPU microbenchmarks
  – Released soon: [github.com/mattsinc/heterosync](https://github.com/mattsinc/heterosync)